Strategies for Service Composition in P2P Networks

Jan Gerke[1], Peter Reichl[2], and Burkhard Stiller[3]

[1] Swiss Federal Institute of Technology ETH Zurich, Computer Engineering and Networks Lab
TIK Gloriastrasse 35, CH-8092 Zurich, Switzerland
gerke@tik.ee.ethz.ch

[2] Telecommunications Research Center FTW Vienna
Donau City Strasse 1, A-1220 Vienna, Austria
reichl@ftw.at

[3] University of Zurich, Department of Informatics IFI and ETH Zrich, TIK
Winterthurerstrasse 190, CH-8057 Zurich, Switzerland
stiller@ifi.unizh.ch

Abstract. Recently, the advance of service-oriented architectures and peer-to-peer networks has lead to the creation of service-oriented peer-to-peer networks, which enable a distributed and decentralized services market. Apart from the usage of single services, this market supports the merging of services into new services, a process called service composition. However, it is argued that for the time being this process can only be carried out by specialized peers, called service composers. This paper describes the new market created by these service composers, and models mathematically building blocks required for such a service composition. A general algorithm for service composition developed can be used independently of solutions for semantic difficulties and interface adaption problems of service composition. In a scenario for buying a distributed computing service, simulated strategies are evaluated according to their scalability and the market welfare they create.

Keywords: Service Composition, Peer-to-Peer Network, Service-oriented Architecture.

1 Introduction

In recent years the development of distributed systems especially the Internet has been influenced heavily by two paradigms: Service-orientation and peer-to-peer (P2P) (A. Oram (Editor), 2001). The benefit of service-oriented architectures (SOA) is their support of loose coupling of software components, *i.e.*, providing a high degree of interoperability and reuse (He, 2003) by standardizing a small set of ubiquitous and self-descriptive interfaces, *e.g.*, the standardization of web services (World Wide Web Consortium, 2004) by the world wide web consortium, and its implementations like .NET (Microsoft Developer Network, 2004). Additionally, P2P file sharing systems like BitTorrent have proven to be scalable content distribution networks (Izal et al., 2004). This scalability is achieved by the distributed nature of P2P networks. Rather than using centralized server infrastructure, they rely on distributed hosts, the peers, and their self-organization abilities. Lately, P2P networks have also been put to commercial use. For instance, Blizzard uses the BitTorrent technology to distribute program updates to their customer base of several hundred thousands, thus reducing the need for a centralized server infrastructure (Blizzard Corporation, 2005).

In order to combine the benefits of the two paradigms and to achieve a decentralized platform for loosely coupled software components, an architecture of a service-oriented middleware was proposed by (Gerke et al., 2003). In order to implement this architecture, (Gerke and Stiller, 2005) presented a JXTA-based middleware. This middleware enables a fully distributed service market, in which peers provide services to other peers. This middleware supports this market by providing mechanisms to search within the underlying distributed P2P network for services, to negotiate the terms of service usage, and finally to charge for service usage. The market itself offers low entrance hurdles, as any Internet host can enter the market by running this middleware implementation.

In addition to the usage of a single service by a single peer, the middleware-enabled service market makes it possible to combine services into new value-added services. This process, called service composition, allows for the maximum benefit from service reusability when creating new services. Thus, new services can be deployed much quicker and the expert knowledge of service developers can be used in new services.

Service composition consists of two main sub problems: (a) understanding what functionality is provided by services and (b) understanding how services communicate through their interfaces. The first problem has been tackled through the development of semantic service descriptions (Hendler et al., 2002), as well as through the application of artificial intelligence (Carman et al., 2003). The second problem is addressed mainly through interface description languages like WSDL (Web Services Descriptions Working Group, 2005). Other interface standards like CORBA do not offer the self-description capabilities of web services, thus reducing the reusability of software components.

The highest benefit from service composition would be achieved, if it could be carried out by any peer in a completely automated manner, at high speed, and costs approaching zero. However, both sub problems described above have not yet been completely solved. Therefore, for the time being, service composition can only be carried out by specialized peers, called service composers at a certain cost.

Therefore, this paper investigates composition strategies for service composers, while abstracting from the detailed technique used. To this end, a general algorithm for service composition is defined and its evaluation is performed in a scenario, where computing power is bought from various peers. Thus, general influences of parameters onto this algorithm are evaluated, and the welfare a service composer can create is defined. The overall goal is to find a strategy for service composition, which scales well while creating high welfare for service consumers and service composers. How this welfare is distributed fairly between the two is out of the scope of this paper, but will addressed elsewhere with the help of public auctions (Varian, 2003).

The remainder of this paper is structured as follows. Section 2 gives an overview of the distributed service market and the underlying P2P architecture. Section 3 contains definitions of services, their properties and descriptions. Section 4 introduces the general service composition algorithm proposed. While Section 5 presents the evaluation of this algorithm with different parameters, Section 6 concludes the work and gives an overview over future work.

2 P2P Service Market

Before investigating service composition, it is necessary to describe the environment in which it takes place. Thus, the underlying service-oriented P2P network and roles being part of the system are outlined. Additionally, the service market enabled by the underlying P2P network and the markets for composed services are discussed.

2.1 Underlying Technology

The P2P network is not a physical network, but is built on top of the Internet (A. Oram (Editor), 2001). This implies that peers are Internet hosts and links of the P2P network are end-to-end (e2e) connections through the Internet between such hosts. Thus, the set of hosts taking part in the P2P network and the e2e connections between them form an overlay network on top of the Internet, consisting of peers and links. In turn, the notion of terms like ’link’ or ’neighbor’ is different in the Internet and in the overlay network. It is assumed that connections provide an e2e quality of service guarantees when required by services, regardless of the technology used to provide these guarantees, *e.g.*, IntServ or DiffServ.

Every peer inside the network can provide and request services to and from other peers. The term service is defined as functionality which is offered by a peer to other peers, and which can be accessed through input and output interfaces. Services are described through documents called *service descriptions*, including service functionality and interfaces, but also characteristics such as terms of usage, *e.g.*, prices and methods of payment. Services can be used by *applications*, which are programs running on peers which are not provided as services themselves and offer user interfaces.

A peer providing a service to another peer is acting as a *service provider*, while a peer which is using a service from another peer is acting as a *service consumer*. A single peer can take over both roles successively or even at the same time, if he provides a set of services to a set of peers and uses a set of other services from another set of peers. The service usage is always initiated by the service consumer, thus a service provider can not supply unrequested services to consumers or even demand payment for services delivered in such a fashion. Due to the dynamic nature of P2P networks, the duration of a service usage is restricted, *i.e.*, it is not possible to rely on the availability of a certain service for weeks or months.

Service usage follows a one-to-one relationship between service consumer and provider, *i.e.*, neither do several service consumers use the same service instance, nor do several service providers together provide a service to a consumer directly. Several service consumers can still use the same service at the same time, but several service instances are created by the service provider and service usage takes place independently. Furthermore, service providers can use services from other service providers, in order to provide a new value-added service to a service consumer. This process of combining services is called *service composition*. A peer carrying out this process is said to play the role of a *service composer*. There is no direct relation between additional service providers and the service consumer. Examples of such service usages are shown in Figure 1.

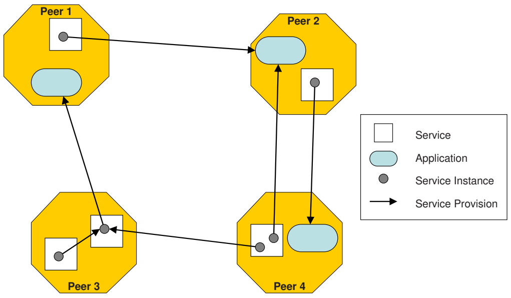

Fig. 1. The use model (Gerke and Stiller, 2005)

2.2 Market Model

The service market consists of the peers which offer services and use them, without composing new services on demand. This service market is assumed to be a global market with low entrance barriers, due to its open P2P nature. This market is characterized through perfect competition between its participants since the number of market participants is large. Therefore, if a peer was offering a service for a price much higher than his own costs, another peer would offer the same service at a lower price. Thus, prices for services are set by the market itself through competition, service prices are given and not negotiable. Service providers do not have to price their services. Either they are able to offer a service for the market price or they do not offer the service at all.

However, the market model is clearly different when service composition is considered. Service composition is a complex process. Therefore, it is assume that only a small number of peers decide to take on the role of service composer. Each of them carries out his own variant of service composition, using his own business secrets. Service composers act as brokers between service consumers and service providers. They take part in the service market as buyers and in the composed service market as sellers, as shown in Figure 2.

Because of the high specialization required, only a small number of overall peers will act as service composer. Therefore, the market for composed services will not have perfect competition but will be dominated by an oligopoly of service composers. In fact, there is a separate market for every new composed service. It is created whenever a service consumer contacts an oligopoly of service composers with the request to compose a new service. Thus, pricing is an important issue in the composed services market, as a price has to be found for a previously inexistent service. Table 1 covers key properties of these two markets.

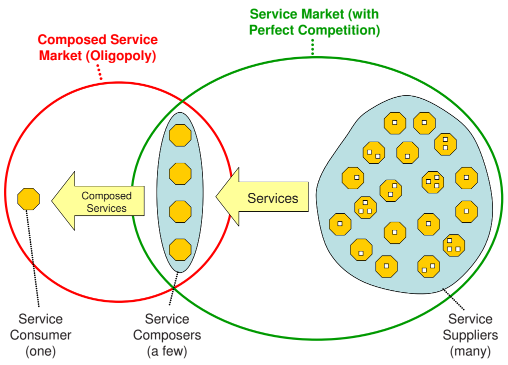
Composed Service Market (Oligopoly)
Service Market (with Perfect Competition)
Composed Services
Services
Service Consumer (one)
Service Composers (a few)
Service Suppliers (many)

Fig. 2. The two markets model

Table 1. Comparison of the Two Markets

	Service Market	Composed Service Market
No. of Sellers	Many	Some
No. of Buyers	Many	One
Market Form	Perfect competition	Oligopoly
Traded Good	Services	Composed Services
Persistence	Constant	Created by request
Prices	Set by market	Pricing required

3 Service Properties

Due to the market-based view and discussion above, goods traded need a formal definition. Therefore, services must be described by specifying their properties as well as the implications of these properties are explained.

(American Heritage Dictionaries (Editor), 2000) defines a service as *an act or a variety of work done for others, especially for pay*. In the scope of this paper, this definition is applied only to the envisioned technical system and its participants (the peers). Thus, a service is a piece of software or a software component operated by one of the systems participants. It fulfills one or several tasks on behalf of another participant of the system, thus carrying out work for him, for which he is paid.

Every service S has properties $S^{'} = (S^{'}_1, ..., S^{'}_n), n \in \mathbf{N}$ which describe the characteristics of the service (Dumas et al., 2001), especially the task which is carried out by the service. These properties may include parameters and information from four different groups:

- Functionality: A formal description of what the service should do (if specified by the service consumer) or what it can do (if specified by the service provider)
- Quality: A list of QoS parameters which describe with which quality the service should be provided (if specified by the service consumer) or can be provided (if specified by the service provider). All parameters can have fixed values or can be described with value ranges.
- Cost: A tariff (Reichl and Stiller, 2001) used to charge the consumption of the service (if specified by the service provider, the actual cost is calculated by applying the tariff to the functionality and quality properties measured during the service delivery) or the willingness-to-pay function (if specified by the service consumer) which describes how much the consumer is willing to pay for a service depending on its functionality and quality parameters.
- Others: Who is providing/requesting the service.

Based on these definitions and explanations of services and their properties the following formalized approach is taken.

Service properties are specified in service descriptions, where every property $S_x^{'}$ consists of a formal *property description* $S_x^{''}$ and a *property range* $S_x^{*} = [\underline{S}_x^{'}, \overline{S}_x^{'}]$. The property description describes the meaning of properties according to a common semantic standard shared by all peers (Hendler et al., 2002). The property range describes the degree of fulfillment of this property in numerical values. Metrics generating these values are part of the property description.

Two property descriptions *match* when they are semantically equal according to the common semantic standard. Thus, a property $A_x^{'}$ is said to *fulfill* another property $B_y^{'}$, if and only if, $A_x^{''}$ matches $B_x^{''}$ and their property ranges overlap, *i.e.*, $\underline{B}_x^{'} \leq \underline{A}_x^{'} \leq \overline{B}_x^{'}$ or $\underline{B}_x^{'} \leq \overline{A}_x^{'} \leq \overline{B}_x^{'}$. Analogously, a property $A_x^{'}$ is said to *exactly fulfill* another property $B_y^{'}$, if and only if, $A_x^{''}$ matches $B_x^{''}$ and $\underline{A}_x^{'} = \underline{B}_x^{'}$ and $\overline{A}_x^{'} = \overline{B}_x^{'}$.

Then, a service S with properties $S^{'} = (S_1^{'}, ..., S_n^{'}), n \in \mathbf{N}$ is said to *implement* a service description D with properties $D^{'} = (D_1^{'}, ..., D_m^{'}), m \in \mathbf{N}$, if and only if, all its properties fulfill the properties specified in the service description, *i.e.*, $n = m$ and $\forall i \in [1, n] : S_i^{'}$ fulfills $D_i^{'}$. Analogously, a service S with properties $S^{'} = (S_1^{'}, ..., S_n^{'}), n \in \mathbf{N}$ is said to *exactly implement* a service description D with properties $D^{'} = (D_1^{'}, ..., D_m^{'}), m \in \mathbf{N}$, if and only if, $n = m$ and $\forall i \in [1, n] : S_i^{'}$ exactly fulfills $D_i^{'}$. Thus, a service S is called an *(exact) implementation* of a service description D, if and only if, S (exactly) implements D. Vice versa, a service description D is said to *(exactly) describe* a service S, if and only if, S (exactly) implements D. It is assumed that exact service descriptions exist and have been published within the P2P network for all services offered by peers acting as service providers. Of course, service descriptions can exist without corresponding implementations.

Even if a consumer specifies fixed property values in a service description, an infinite number of different services can implement his service description. Thus, a *service class* is defined as a set of services which has specific common properties. Let $S = \{S_1, ..., S_n\}, n \in \mathbf{N}$ be the set containing all services and let D be a service description with properties $D^{'} = (D_1^{'}, ..., D_m^{'}), m \in \mathbf{N}$. Then, $\widehat{D}$ is called a service class for a service description D, if and only if, $\widehat{D} = \{S_x | S_x \text{ implements } D\}, S_x \in S$. All services which have these properties are called *members* of the service class. Let C be the service class for a service description D. Then, a service S is called a *member* of C, if and only if, S implements D. Thus, every implementation of a service description is a member of the service class created by the description. Furthermore, every service description automatically creates a service class, though this class does not need to have any members.

4 Service Composition

From the service composers point of view, the service composition process starts when he receives a service description from a service consumer. The service consumer does not have to state fixed service properties within this description, but can make use of property ranges as defined in Section 3.

After receiving the consumers service description, the composer searches the P2P network for member services of the service class described by this description. If he can not find such a service, he starts composing the service by combining other services. In order to do this, he must obtain or create a building plan (Gerke, 2004), which describes how to achieve the properties of the original service class by combining members from other service classes. Generally, a building plan describes how members of a specific service class can be combined with members of other specific service classes to create a new composed service. In particular, it describes properties of the composed service and how these properties relate to the properties of its component services. This process is supported by the mapping m in such a way that if D is a service with properties $D^{'}$, then B is called a building plan for D, if and only if, B contains service descriptions $\{D_1, ..., D_n\}, n \in (N)$ and a mapping $m : D_1^{'} \times ... \times D_n^{'} \rightarrow D^{'}$ which describes the relation between properties of component services (*i.e.*, specific services fulfilling the service descriptions) on the properties of the composed service (*i.e.*, the service created by composing the component service as also described in the building plan).

When the composer has obtained one or several building plans for the service class described by the consumers service description, he searches the P2P network for implementations of the service classes described within them. If he is unable to find any of these service classes, he recursively creates building plans for this service class. Thus, he creates a service dependency graph, as shown in Figure 3.

The original service class (described by the consumers service description) is the root of this tree. Several ways to implement this service (one implementation and two building plans) are connected to the root node via an or node, while service classes are connected to the building plan via and nodes. Composed services can be found by traversing the tree from its root node. At each or node exactly one link must be chosen for traversal, while at each and node all links have to be traversed. A sub tree found by applying this algorithm models a composed service when only services form this trees leaves. Such a tree is called a *cover* of the original service dependency tree and is also shown in Figure 3. The algorithm does rate the quality of a composed service in any way. Building plans only ensure that every composed service fulfills the consumers initial service description (*i.e.*, the functionality is the same and all properties overlap with described properties).

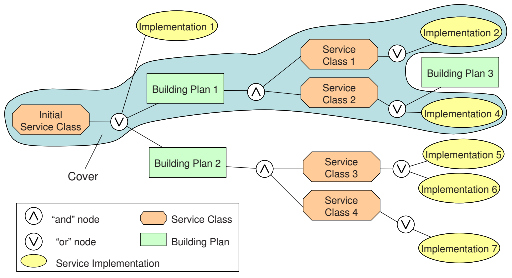

Fig. 3. An example service dependence tree

The following pseudo code defined a recursive algorithm to build a service dependence tree.

```
buildDependenceTree(ServDescr SC) {
   while (ServDescr SI =
   searchNewServiceImplementation (SC)
   != nil) {
      buildDependenceTree (SI);
      SC.addChild (SI);
   }
   while (BuildingPlan BP =
   searchNewBuildingPlan (SC) != nil) {
      for (int i=0; i <
      BP.getChildren().length; i++)
          buildDependenceTree
          (BP.getChildren()[i];
       SC.addChild (BP);
   }
}
```

In order to achieve this presentation, slightly simplified, the following assumptions have been made:

- The classes ServiceDescription and BuildingPlan are tree nodes.
- The and and or nodes are neglected. All children of a ServiceDescription are inherently connected via an or node and all children of a BuildingPlan are inherently connected via an and node.
- Each new BuildingPlan already comes with ServiceClass children (instead of connecting them by hand according to service descriptions).

The algorithm describes a general way to build service dependence trees. By using it, any kind of service dependence tree can be created. The decision about what tree is created is made by the two functions searchNewServiceImplementation and searchNewBuildingPlan. These methods include individual secrets of each service composer, namely:

- How to find a new building plan or service implementation for a given service description.
- When to stop searching for a new building plan or service implementation.

5 Evaluation

The general service algorithm described in Section 4 has been evaluated by simulations. The service requested in these simulations is a computing service with a computing power measured in GFLOPS (billions of floating point operations per second). Different strategies for when to stop searching for more building plans are evaluated. The criteria for evaluation is the achieved average welfare $W(S)$ of a composed service S. The welfare of a composed service depends on the utility $U(S)$ the consumer receives from the service, the cost of the service $C(S)$, and the cost for composing the service $B(S)$ in such a way that $W(S) = U(S) - C(S) - B(S)$ holds. The cost of the composed service is equal to the sum of the costs of the services needed to create it. *I.e.*, let $S_1 ... S_i, i \in [1, m]$ be the services used to create the composed service S, then $C(S) = \sum_{i=1}^{m} C(S_i)$.

5.1 General Assumptions

The simulations make the following assumptions:

- The code to be executed with the computing power can be parallelized to a granularity of 1 GFLOPS, which is also the smallest amount which can be bought as a single service.
- The parallelization incurs no extra effort. Especially, no additional synchronization or signalling overhead must be taken into account.
- Any amount of required GFLOPS can be bought directly as a service implementation from at least one service provider. This assumption is reasonable due to the perfect competition existing in the service market.
- For any amount of GFLOPS five building plans exist, which describe how to obtain the GFLOPS by buying two GFLOPS shares instead. This assumption is not stringent, as there are not many differences between finding a bad building plan and finding none at all. Plans describe 0.9/0.1 (worst), 0.8/0.2, 0.7/0.3, 0.6/0.4 and 0.5/0.5 (best) ratios of the original amount of GFLOPS.
- The cost of finding a new building plan or finding out that no more building plan exist is fixed and is equal to . However, it is also assumed that the service composer has no previous knowledge of the cost, *i.e.*, only after every search for building plan does he know what this search did cost.
- The cost of finding service implementations is negligible, as this functionality is provided by the service search of the underlying P2P network. Thus, let n be the number of searched building plans, then $B(S) = n \cdot Z$.
- The utility function of the service consumer is increasing monotonously and it is convex, based on the amount of GFLOPS received. Thus, additional utility created by additional amounts of GFLOPS decreases monotonously.

- Since there is perfect competition in the service market, all service providers use a similar cost function. This cost function is increasing monotonously and it is concave, based on the amount of GFLOPS received. Thus, the cost of additional GFLOPS is monotonously increasing.
- Properties are fixed. This means, that the service consumer does not specify a range of GFLOPS, but rather a precise amount of GFLOPS. Since building plans define exact ratios for their two respective component services, the amount of GFLOPS in service classes and service implementations also becomes fixed.

5.2 Simulation Setup

The service composition algorithm was implemented in Java and executed with different search strategies and parameters. Simulations were carried out on a PC with 512 MB memory and a 1.8 GHz CPU. For all settings, the average welfare was calculated as the average over 100 simulation runs. In all simulations the amount of the requested computing power has been set to 1024 GFLOPS. The cost function used in those simulations is $C(x) = \frac{x^2}{10000}$, the utility function is $U(x) = \log x \cdot 10$ (x being the number of GFLOPS), as shown in Figure 4.

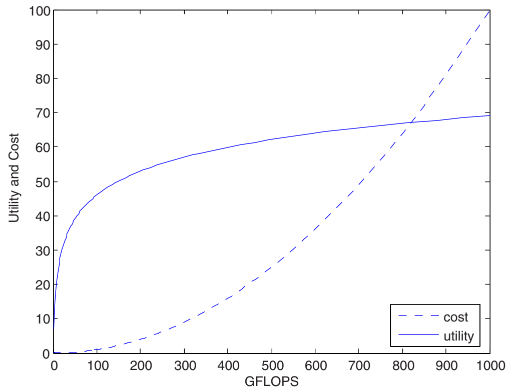

Fig. 4. Utility and cost functions

5.3 Balanced Search

The algorithm used to build the service dependence tree is constructed from the root as a balanced tree. This means that for every service class in the tree the same amount of searches for building plans is carried out until the previously defined search depth is reached. Figure 5 depicts the welfare for different costs of finding building plans, when building plans are only searched once, *i.e.*, no recursion takes place (strategy #1). Vice versa, Figure 6 depicts the welfare for different costs of finding building plans, when recursion takes place until a certain tree depth is reached. At all steps only a single building plan is searched (strategy #2).

Both strategies produce good results for the average cost for searching building plans of 3. Thus, this cost was chosen for strategy #3, which investigated, whether the welfare

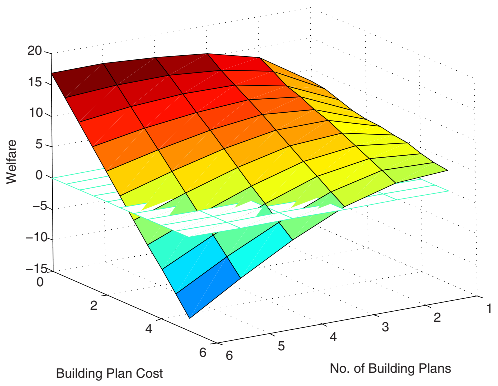

Fig. 5. Welfare for different numbers of building plan searches and different building plan costs

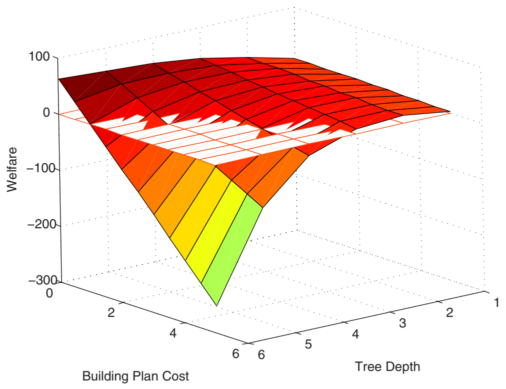

Fig. 6. Welfare for different service dependency tree depths and different building plan search costs

could be increased by varying the number of building plans searched at every step, as well as the tree depth. However, the results depicted in Figure 7 clearly show, that the increased benefit achieved by building a wide and deep service dependency tree is by far outweighed by the increased cost for building this tree. In order to optimize the welfare, more sophisticated heuristics for influencing the trees width and depth are required.

5.4 Search Until Welfare Decreases

In order to avoid search cost explosions (cf. Figure 7), the service composition algorithm was extended with strategy #4: The service dependency tree is built recursively without a predefined search depth. The recursion is stopped when the previous search for a building plan has created less additional benefit than the search cost, *i.e.*, the welfare decreased.

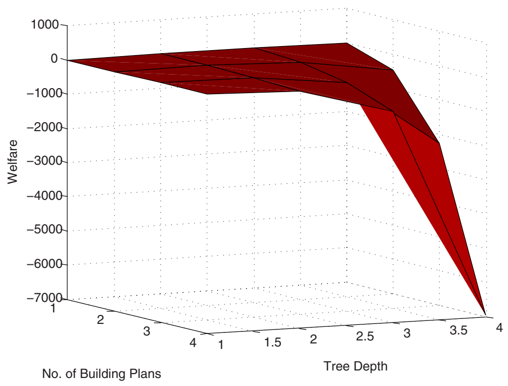

Fig. 7. Average welfare for different numbers of building plan searches and different service dependency tree depths, with fixed building plan cost of 3

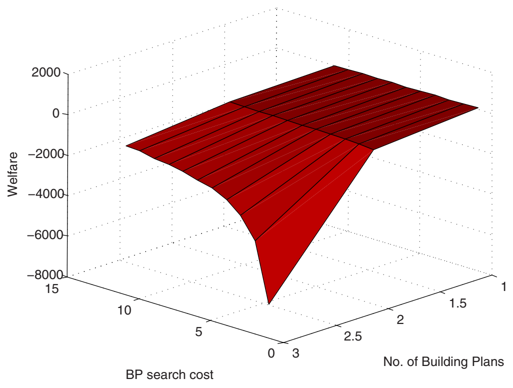

Fig. 8. Average welfare for different numbers of building plan searches and building plan search costs, using the local welfare decrease heuristic

Strategy #4 is clearly not optimal, as the next search for a building plan could still provide a higher benefit. However, results depicted in Figure 8 clearly show that this strategy makes the algorithm resistant against high building plan search costs. Still, the algorithm produces bad results for higher numbers of building plans searched.

5.5 Tree Width Decreases with Increasing Tree Depth

Strategy #5 makes the following extension to the previous one: If the search continues (*i.e.*, the last search step produced additional welfare), the number of building plans searched will be decreased by one in comparison to the previous search step. This strategy outperforms the previous one in all cases except one: When initially only a single building plan is searched the tree depth will only be 1 and the resulting welfare will be equal to the welfare as depicted in Figure 9 for the search of a single building plan.

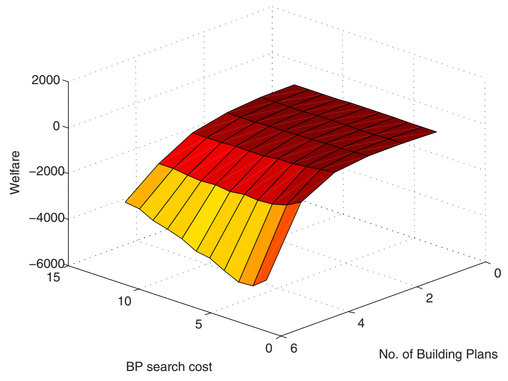

Fig. 9. Average welfare for strategy #5

5.6 Tree Width Decreasing with Welfare Decrease

Finally, strategy #6 computes the number of building plans depending on the size of the additional welfare created in the last search step. The welfare divided by the cost for searching the building plan equals the number of building plans to be searched next. This strategy performs worse than the previous one for higher numbers of initially searched building plans, but always creates higher welfare for a maximum number of searched building plans of 1.

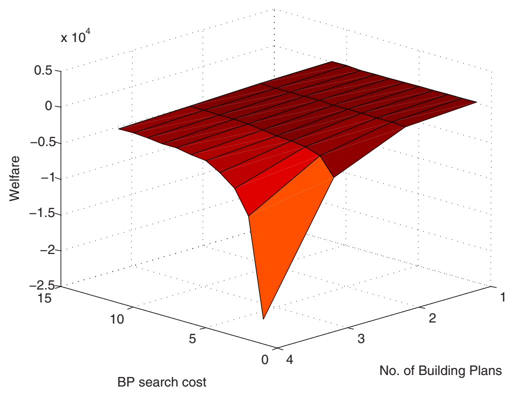

Fig. 10. Average welfare for strategy #6

5.7 Discussion

The set of strategies show a number of different heuristics. However, measuring scalability requires to count the number of searches for building plans carried out, which have been omitted in detail for space reasons, but they are part of the calculated welfare in the form of the cost for building the service dependence tree. Thus, the depicted welfare functions clearly show the influence of bad scalability in the rapid drops of welfare due to a high number of searches for building plans.

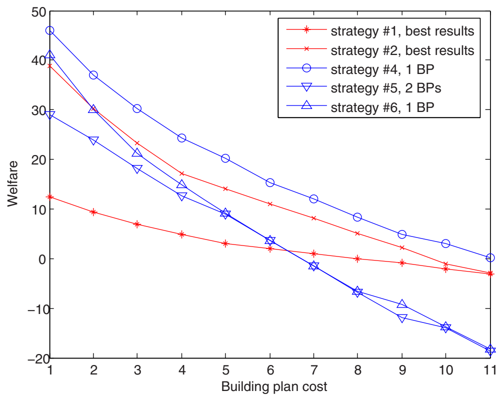

Fig. 11. Comparison of optimal sub-strategies

On one hand, simulation results show that all strategies produce bad results when building a wide service dependence tree, *i.e.*, when searching for several building plans for a single service class. On the other hand, results indicate that controlling the depth of the tree, *i.e.*, deciding when to stop the recursion, can be handled very well by strategies. For each of theses strategies an optimal sub-strategy exists. Strategy #4 and #6, produce their highest welfare when at each algorithm step a single building plan at the most is searched. Strategy #5, produces its highest welfare when initially searching for two building plans. Welfare of these sub strategies are compared in Figure 11. It can be clearly seen, that strategy #4 outperforms all other strategies. Furthermore, it produces higher welfare than the theoretical hulls of strategies #1 and #2, which were created by manually selecting the best sub strategy for each building plan search cost. Thus, for any building plan search cost investigated the described sub strategy #4 is the best choice. Furthermore, the results depicted in Figure 11 indicate that for high building plan search costs strategy #4 produces the same welfare as the hulls of strategy #1 and #2. This is logical, since for high building plan search costs no profit can be made, in which case all three strategies will stop after a single search for a building plan.

6 Summary

Different strategies for service composition in P2P networks have been investigated. Based on the distributed and decentralized service market together with its underlying P2P network, the service market model was extended by the composed service market. Driven by models of all building blocks for the service composition process the generic service composition algorithm was specified, resulting in service dependence trees. Different strategies for building this tree were developed and evaluated in simulations of composing a distributed computing service.

Such strategies for building the service dependence tree can differ in the width of the tree (*i.e.*, how many building plans are searched for each service class) and the depth of the tree (*i.e.*, when to stop the recursion of the tree building algorithm). Those results show that extending the tree and the width at the same time quickly leads to a dropping welfare of several orders of magnitude. However, improved heuristically strategies control the width and depth of the tree. All strategies proved to be very sensitive to the number of building plans searched at each step. Not a single heuristic was able to produce a high (or even positive) welfare, when a larger number of building plans could be searched for. In fact, these results indicate that at the most two building plans should be searched for within any service class. If those strategies were used in this way, they produced positive welfare for most of the investigated spectrum of building plan search costs. One strategy produced positive welfare for the whole spectrum, and continually outperformed all other strategies.

Thus, it has been shown that service composition can be economically successful, when a generic algorithm is used with the proposed strategies and a small number of building plans searched for at each step. Future work includes the development of public auctions for composed services, thus enabling the service consumer to publish his utility function, which in turn enables the maximization of the welfare of the composed service market.

This work has been partially performed in the EU-projects MMAPPS and DAIDALOS, and the Austrian Kplus competence center programme.

References

Oram, A. (ed.): Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly (2001)

American Heritage Dictionaries (Editor): The American Heritage Dictionary of the English Language, 4th edn. Houghton Mifflin (2000)

Blizzard Corporation, Blizzard Downloader F.A.Q. (2005), `http://www.worldofwarcraft.com/info/faq/blizzarddownloader.html`

Carman, M., Serafini, L., Traverso, P.: Web service composition as planning. In: ICAPS 2003 Workshop on Planning for Web Services, Trento, Italy (2003)

Dumas, M., O'Sullivan, J., Heravizadeh, M., ter Hofstede, A., Edmond, D.: Towards a semantic framework for service description. In: IFIP Conference on Database Semantics, Hong Kong, China (2001)

Gerke, J.: Service composition in peer-to-peer systems. In: Proceedings of the Dagstuhl Seminar on Peer-to-Peer Systems and Applications, Dagstuhl, Germany (2004)

Gerke, J., Hausheer, D., Mischke, J., Stiller, B.: An architecture for a serviceoriented peer-to-peer system (sopps). In: Praxis der Informationsverarbeitung und Kommunikation (PIK), 2 (2003)

Gerke, J., Stiller, B.: A service-oriented peer-to-peer middleware. In: Müller, P., Grotzhein, R., Schmitt, J.B. (eds.) Kommunikation in Verteilten Systemen (KiVS) 2005, Informatik Aktuell. Springer, Heidelberg (2005)

He, H.: What is Service-Oriented Architecture. In: O'Reilly webservices.xml.com (2003), `http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html`

Hendler, J., Berners-Lee, T., Miller, E.: Integrating applications on the semantic web. Journal of the Institute of Electrical Engineers of Japan 122(10) (2002)

Izal, M., Urvoy-Keller, G., Biersack, E., Felber, P., Hamra, A.A., Garces-Erice, L.: Dissecting bittorrent: Five months in a torrent's lifetime. In: Passive and Active Measurements (PAM), Antibes Juan-les-Pins, France (2004)

Microsoft Developer Network, NET Framework Product Overview (2004), http://msdn.microsoft.com/netframework/technologyinfo/overview/

Reichl, P., Stiller, B.: Nil nove sub sole? why internet tariff schemes look like as they do. In: 4th Internet Economics Workshop (IEW'01), Berlin, Germany (2001)

Varian, H.: Intermediate Microeconomics: A Modern Approach. W.W. & Company, Norton (2003)

Web Services Descriptions Working Group, Web Services Description Language (WSDL) Version 2.0. (2005), http://www.w3.org/2002/ws/desc

World Wide Web Consortium, Web Services Activity Statement (2004), http://www.w3c.org/2002/ws/Activity